**Description of System 4**: We are looking for a system for an autonomic vehicle. The system will monitor the vehicle at all times and it cannot make errors. The security of the passengers is of utmost importance. The system should function for any vehicle.

**Functional Requirement:**

1. The system shall continuously monitor the vehicle's status at all times.
2. The system shall detect and respond to any changes in the vehicle's operation in real-time.
3. The system shall ensure the security of all passengers during vehicle operation.
4. The system shall operate without making any errors during the monitoring process.
5. The system shall be able to function across various vehicle types, including cars, trucks, and buses.

**Non-Functional Requirements:**

1. The system shall have a high availability rate of 99.999% to ensure continuous monitoring.
2. The system shall have a maximum response time of 100 milliseconds for detecting and reacting to vehicle changes.
3. The system shall be scalable to accommodate various vehicle types without performance degradation.
4. The system shall be secure against cyberattacks and unauthorized access, ensuring passenger safety at all times.
5. The system shall be compliant with international safety and vehicle standards across different regions.
6. The system shall be able to operate under different environmental conditions, including extreme temperatures, humidity, and rough terrain.
7. The system shall consume minimal power to avoid excessive energy usage in the vehicle.
8. The system shall have a user-friendly interface for operators or engineers performing maintenance and diagnostics.

**Use case Scenarios:**

**Use Case 1: Continuous Monitoring of the Vehicle**

* **Primary Actor:** Autonomic Vehicle System
* **Precondition:** The vehicle is in operation.
* **Trigger:** The vehicle starts, and the system is activated.
* **Main Success Scenario:**
  1. The system initiates continuous monitoring as soon as the vehicle is operational.
  2. The system collects real-time data from the vehicle’s sensors.
  3. The system analyzes the data to detect any anomalies or issues.
  4. The system maintains continuous monitoring without interruptions.
* **Postcondition:** The vehicle is continuously monitored without errors during the entire operation.

**Use Case 2: Real-Time Response to a Critical Issue**

* **Primary Actor:** Autonomic Vehicle System
* **Precondition:** The system is actively monitoring the vehicle.
* **Trigger:** A critical issue is detected, such as a brake failure.
* **Main Success Scenario:**
  1. The system detects the brake failure in real time.
  2. The system alerts the driver and passengers of the issue.
  3. The system automatically activates safety protocols (e.g., slowing the vehicle, engaging emergency brakes).
  4. The system continuously monitors the situation until the issue is resolved or the vehicle is safely stopped.
* **Postcondition:** The issue is addressed with no harm to the passengers, and the system returns to its normal monitoring state.

**Use Case 3: Ensuring Passenger Security**

* **Primary Actor:** Autonomic Vehicle System
* **Precondition:** Passengers are in the vehicle, and the system is active.
* **Trigger:** A potential security threat is detected (e.g., unauthorized access to the vehicle).
* **Main Success Scenario:**
  1. The system detects unauthorized access or a breach attempt.
  2. The system immediately locks down the vehicle and notifies the passengers.
  3. The system alerts emergency services and provides real-time location data.
  4. The system continues monitoring until the threat is neutralized.
* **Postcondition:** The passengers remain safe, and the security threat is mitigated.

**Use Case 4: System Adaptation Across Different Vehicle Types**

* **Primary Actor:** Autonomic Vehicle System
* **Precondition:** The system is installed in a new vehicle type (e.g., a truck).
* **Trigger:** The vehicle is turned on, and the system starts monitoring.
* **Main Success Scenario:**
  1. The system detects the new vehicle type and adapts its monitoring protocols accordingly.
  2. The system begins continuous monitoring specific to the truck's operational requirements.
  3. The system ensures that all functionalities are appropriately tailored to the vehicle type without degradation in performance.
* **Postcondition:** The system successfully adapts and monitors the new vehicle type without errors.

**Use Case 5: Error-Free Operation During Monitoring**

* **Primary Actor:** Autonomic Vehicle System
* **Precondition:** The system is actively monitoring the vehicle.
* **Trigger:** Regular vehicle operation continues, and the system is running.
* **Main Success Scenario:**
  1. The system performs continuous error-free monitoring throughout the vehicle's journey.
  2. The system handles all data inputs and outputs without experiencing failures or downtime.
  3. The system logs all activities for later review and diagnostics without any errors.
* **Postcondition:** The vehicle completes its operation with the system maintaining error-free functionality.

**Domain concepts:**

**1. Vehicle Monitoring**

* **Sensors:** Devices used to continuously gather data from various parts of the vehicle (e.g., speed, engine temperature, fuel level).
* **Telemetry Data:** Real-time data collected from sensors within the vehicle.
* **Monitoring Algorithms:** Software that processes sensor data and detects anomalies or potential issues.
* **Diagnostic System:** A component responsible for analyzing the vehicle's condition and providing reports on system health.

**2. Real-Time Processing**

* **Real-Time Detection:** The ability of the system to detect changes or issues in the vehicle’s operation as they happen.
* **Response Protocols:** Predefined actions that the system takes in response to specific events (e.g., brake failure, engine malfunction).
* **Event Handling:** Mechanisms that manage system responses to detected issues, including prioritization and execution of safety protocols.

**3. Passenger Security**

* **Passenger Identification:** The process of verifying and tracking the identity of passengers within the vehicle.
* **Access Control:** Security measures that prevent unauthorized access to the vehicle.
* **Threat Detection:** Systems that identify potential security threats (e.g., intrusion detection, abnormal behavior analysis).
* **Emergency Protocols:** Actions taken by the system to protect passengers during a security threat (e.g., lockdown procedures, communication with authorities).

**4. Error-Free Operation**

* **Fault Tolerance:** The system’s ability to continue functioning correctly even if some of its components fail.
* **Self-Diagnosis:** The system’s ability to detect and report its own faults or malfunctions.
* **Error Handling:** Mechanisms to manage and recover from errors without impacting the vehicle’s operation.

**5. Adaptability Across Vehicle Types**

* **Vehicle Type Profiles:** Predefined configurations that allow the system to adapt to different vehicle types (e.g., cars, trucks, buses).
* **Configuration Management:** The process of adjusting system parameters to match the requirements of the specific vehicle type.
* **Scalability:** The ability of the system to function efficiently across vehicles of different sizes and capabilities without performance degradation.

**6. User Interface**

* **Operator Interface:** The part of the system that allows engineers or operators to interact with the system for maintenance, diagnostics, and configuration.
* **Dashboard:** A graphical display that provides real-time information about the vehicle's status to the operator or driver.
* **Alerts and Notifications:** Mechanisms that inform users (passengers, drivers, or operators) of critical system events or conditions.

**7. Environmental Adaptation**

* **Environmental Sensors:** Devices that detect external conditions such as temperature, humidity, and road conditions.
* **Climate Control Adjustments:** The system's capability to adjust vehicle operations based on environmental inputs (e.g., extreme weather conditions).
* **Ruggedization:** Design aspects that ensure the system remains operational under harsh environmental conditions.

**8. Cybersecurity**

* **Encryption:** Techniques used to secure communication between the vehicle system and external entities.
* **Authentication Mechanisms:** Processes for verifying the identity of users and preventing unauthorized access to the system.
* **Intrusion Detection System (IDS):** A component responsible for identifying and preventing cyberattacks on the vehicle’s systems.

**9. Safety Standards Compliance**

* **Regulatory Compliance:** Ensuring that the system adheres to relevant safety and operational standards (e.g., ISO 26262 for road vehicle functional safety).
* **Certification Processes:** The steps required to certify that the system meets industry and governmental safety requirements.
* **Safety Protocols:** Predefined processes that ensure passenger and vehicle safety in all scenarios.

**10. Power Management**

* **Power Consumption:** The amount of energy the system uses during operation.
* **Energy Efficiency:** Optimization techniques to minimize power usage without compromising performance.
* **Battery Management:** Ensuring the system operates effectively while managing the vehicle’s battery resources efficiently.

**11. Logging and Data Storage**

* **Event Logging:** Recording events, errors, and system actions for auditing and diagnostics purposes.
* **Data Retention:** Policies for storing historical data over a certain period.
* **Cloud Integration:** The ability to sync data with external cloud systems for backup and extended processing.

**Domain** **Model:**

Picture attached domain\_model\_4

**Suggested Architectural Style**: Microservices Architecture

Given the complexity and scalability requirements of the autonomic vehicle system, a **Microservices Architecture** would be well-suited for this domain model. Here's why:

1. **Decoupling of Components:** Each domain concept, such as "Vehicle Monitoring," "Real-Time Processing," "Passenger Security," and others, can be implemented as independent services. This allows for easier scaling, development, and maintenance.
2. **Fault Isolation:** If one service fails (e.g., Passenger Security), the rest of the system can continue operating, ensuring high availability and fault tolerance.
3. **Scalability:** Services can be scaled independently based on demand. For example, real-time processing services may need more resources than data storage services.
4. **Technology Flexibility:** Different services can be built using different technologies, allowing for the best tools to be chosen for each domain concept.
5. **Security and Compliance:** Sensitive areas, like cybersecurity, can be isolated in their own services, ensuring that specific security protocols are followed.

Diagram attached as component\_diagram\_4\_1

**Justification for the architecture:**

Here’s a detailed justification for using the **Microservices Architecture** to implement the autonomic vehicle system:

**1. Modularity and Separation of Concerns**

* **Justification:** Microservices architecture promotes separation of concerns by breaking down the system into independent, loosely coupled services. Each service is responsible for a specific domain concept (e.g., Vehicle Monitoring, Passenger Security). This allows development teams to focus on individual services without worrying about the entire system, making the system easier to understand, develop, and maintain.
* **Benefit:** This modularity aligns well with the complexity of the system, where different functionalities like monitoring, security, and adaptability need to operate independently yet cohesively.

**2. Scalability**

* **Justification:** Different components of the system will have varying resource demands. For example, the real-time processing and monitoring services might need to scale more aggressively than the logging or user interface services. Microservices allow each service to scale independently based on its specific load and performance requirements.
* **Benefit:** This enables the system to handle a wide range of vehicle types and operational environments without performance bottlenecks, meeting the non-functional requirement for scalability.

**3. Fault Isolation and Resilience**

* **Justification:** In a microservices architecture, each service operates independently. If one service fails (e.g., Threat Detection), it does not necessarily cause the entire system to fail. Fault isolation ensures that the rest of the system can continue to function while the failing service is being recovered.
* **Benefit:** This enhances the overall reliability of the system, which is critical for an autonomic vehicle system that must prioritize safety and error-free operation.

**4. Technology Agnosticism**

* **Justification:** Microservices architecture allows each service to be developed and deployed using the most appropriate technology for its specific needs. For instance, the real-time processing service might require a high-performance language like C++, while the logging service could be built using a language more suitable for data handling, like Python.
* **Benefit:** This flexibility ensures that the best tools and frameworks are used for each domain concept, optimizing performance and maintainability.

**5. Continuous Deployment and Independent Development**

* **Justification:** With microservices, development teams can work on different services in parallel, without waiting for other teams to complete their tasks. Additionally, services can be deployed independently, allowing for continuous deployment and quick iterations.
* **Benefit:** This accelerates development cycles, which is essential in a system that needs to adapt to different vehicle types and evolving safety standards. Continuous updates to specific services can be deployed without taking down the entire system.

**6. Scalability and Cloud Readiness**

* **Justification:** Microservices are inherently cloud-friendly, allowing services to be distributed across cloud environments, which can automatically scale them as needed. The ability to deploy individual services in the cloud makes it easier to achieve global scalability and resilience.
* **Benefit:** This meets the non-functional requirement for high availability and ensures that the system can handle the real-time demands of vehicle monitoring and security, even in large-scale deployments.

**7. Enhanced Security**

* **Justification:** Security can be managed at the service level. For instance, the Cybersecurity Service can be isolated from other services, ensuring that security protocols, encryption, and access controls are enforced strictly without interfering with other system operations.
* **Benefit:** This isolation strengthens the system's overall security, which is critical given that passenger safety and preventing unauthorized access are top priorities.

**8. Compliance and Adaptability**

* **Justification:** With microservices, services like Safety Compliance and Adaptability can be updated and re-certified independently of the rest of the system. This makes it easier to adapt to new regulations or support different vehicle types without requiring extensive changes to the entire architecture.
* **Benefit:** This flexibility ensures that the system remains compliant with evolving safety standards and can adapt to a wide range of vehicle types, addressing both functional and non-functional requirements.

**Conclusion**

Microservices architecture offers the right balance of modularity, scalability, resilience, security, and adaptability. It aligns with the system's requirements for continuous monitoring, error-free operation, and passenger security across various vehicle types. This architecture also supports independent scaling, development, and deployment of services, making it ideal for handling the complex and dynamic nature of the autonomic vehicle system.

**Suggested Architectural Style**: Event Driven Architecture

Another suitable architectural style for this autonomic vehicle system is **Event-Driven Architecture**. This approach focuses on the production, detection, and consumption of events. It’s ideal for systems that require real-time processing and responses, such as an autonomic vehicle monitoring system.

**Reasons for Choosing Event-Driven Architecture:**

1. **Real-Time Responsiveness:** This architecture excels in environments where real-time detection and response are crucial. The system can react to events as they happen, which aligns well with the system’s functional requirement of continuous monitoring and error-free operation.
2. **Loose Coupling:** Event-driven systems allow for components to be loosely coupled, where services only need to know about the events they are interested in, rather than the entire system.
3. **Scalability:** Components can be scaled independently based on the volume of events they need to process. For example, real-time processing services can scale as the number of vehicle events increases.
4. **Fault Tolerance:** If one event handler fails, other components are unaffected, which increases system resilience. This fault isolation is critical for ensuring the safety of passengers.

Diagram attached as component\_diagram\_4\_2

**Justification for the architecture:**

Here’s a detailed justification for using the **Event-Driven Architecture** to implement the autonomic vehicle system:

**1. Real-Time Responsiveness**

* **Justification:** Event-driven architecture is designed to respond to events as they occur, making it ideal for systems that require real-time detection and immediate action. For the autonomic vehicle system, events like sensor data changes, security threats, or system faults need to be processed in real-time to ensure passenger safety and vehicle functionality.
* **Benefit:** This architecture aligns perfectly with the system's requirement for continuous monitoring and error-free operation by allowing the system to respond immediately to critical events.

**2. Loose Coupling**

* **Justification:** In event-driven systems, services communicate through events rather than direct calls. This means that services are decoupled and don’t need to know about each other's internal workings, only the events they produce or consume. This leads to greater flexibility in how services are developed, deployed, and updated.
* **Benefit:** Loose coupling makes it easier to develop and maintain the system, especially as new vehicle types or features are added. It also reduces the impact of changes, allowing for independent updates to services without affecting the whole system.

**3. Scalability**

* **Justification:** Event-driven architecture naturally supports scalability, as services can independently scale based on the volume of events they need to process. For example, as more vehicles are added to the system, the services handling real-time data processing or logging can be scaled horizontally to handle the increased event traffic.
* **Benefit:** This scalability is crucial for a system that must monitor a wide range of vehicles in real time and ensure high availability and responsiveness.

**4. Fault Tolerance and Resilience**

* **Justification:** In an event-driven system, failure in one service does not necessarily bring down the entire system. If a service fails to process an event, the event can be retried or redirected to a different service instance. This fault isolation is critical for ensuring that the autonomic vehicle system remains operational even if individual components fail.
* **Benefit:** This architecture ensures high reliability, which is essential for maintaining passenger safety and the continuous monitoring of the vehicle.

**5. Flexibility for Complex Interactions**

* **Justification:** Event-driven architecture is well-suited for handling complex interactions where many different services may need to respond to the same event. For example, a single event indicating a security threat could trigger multiple services to respond, such as locking the vehicle, notifying emergency services, and alerting passengers.
* **Benefit:** This flexibility allows the system to handle intricate and interdependent responses to events, which is essential for scenarios involving passenger security and vehicle safety.

**6. Event-Based Security Handling**

* **Justification:** Security events, such as unauthorized access attempts, can trigger immediate system responses in an event-driven architecture. This architecture allows security-related events to propagate throughout the system instantly, enabling multiple layers of security to act concurrently (e.g., locking doors, alerting authorities, and activating alarms).
* **Benefit:** Event-driven systems enhance the security response capabilities of the autonomic vehicle system by allowing for quick and distributed handling of security events, thereby protecting passengers and the vehicle.

**7. Asynchronous Processing**

* **Justification:** Event-driven systems are inherently asynchronous, meaning that services do not need to wait for each other to complete processing. This is particularly important in real-time systems where latency could be a critical issue. For instance, telemetry data from the vehicle's sensors can be processed without waiting for responses from other services, ensuring that the system remains responsive.
* **Benefit:** Asynchronous processing ensures that the system remains responsive and can handle a high volume of events without becoming bottlenecked by slower operations.

**8. Simplified Event Logging and Auditing**

* **Justification:** In an event-driven architecture, every action and state change is triggered by an event. This creates a natural logging mechanism where events can be easily recorded, tracked, and audited. This is particularly useful for the logging and compliance requirements of the system, allowing for easy tracing of what happened, when, and why.
* **Benefit:** This inherent event logging simplifies the implementation of data retention, regulatory compliance, and diagnostics, making it easier to ensure that the system meets safety and security standards.

**9. Easy Integration with Cloud Services**

* **Justification:** Event-driven architectures are well-suited for integration with cloud services and can take advantage of cloud-based event buses and message brokers (e.g., AWS EventBridge, Azure Event Grid). This allows for distributed processing and easy scaling across multiple regions.
* **Benefit:** By leveraging cloud infrastructure, the system can easily scale to meet global demands, handle large volumes of events, and ensure that the system remains resilient and available across different regions and vehicle types.

**Conclusion**

**Event-Driven Architecture** is an excellent choice for the autonomic vehicle system because of its focus on real-time responsiveness, loose coupling, scalability, fault tolerance, and flexibility. These characteristics make it well-suited for a system that must continuously monitor vehicles, respond to security threats, and handle complex interactions in real-time across various vehicle types. The ability to scale and handle asynchronous processing ensures that the system can grow and adapt to new requirements, making it a robust solution for this domain.

**Software Architecture Analysis Method (SAAM) for Event Driven Architecture**

To apply the **Software Architecture Analysis Method (SAAM)** to the event-driven architecture proposed for the autonomic vehicle system, we need to identify and document system scenarios. SAAM helps evaluate the quality of a software architecture by focusing on various scenarios, including functional scenarios (which describe the system's expected behavior) and non-functional scenarios (which cover quality attributes like performance, scalability, and security).

**Step 1: Define Key Scenarios**

Here are several scenarios that address both functional and non-functional aspects of the event-driven architecture for the autonomic vehicle system.

**1. Functional Scenarios**

**Scenario 1: Real-Time Vehicle Monitoring**

* **Description:** The system continuously monitors the vehicle’s sensor data (e.g., speed, engine temperature) and triggers events when specific thresholds are exceeded.
* **Expected Behavior:** The Vehicle Monitoring Service captures real-time telemetry data and sends events to the Event Bus. Other services (e.g., Real-Time Processing Service) react to these events, analyzing the data and triggering responses as necessary.
* **Evaluation:** This scenario tests the core functionality of real-time data monitoring and processing. The event-driven architecture's ability to handle asynchronous data collection and dispatching is critical here.

**Scenario 2: Security Threat Detection and Response**

* **Description:** An unauthorized attempt to access the vehicle is detected. The system must respond immediately by locking down the vehicle, notifying emergency services, and alerting passengers.
* **Expected Behavior:** The Threat Detection Service sends an event to the Event Bus. The Passenger Security Service responds by locking the vehicle and triggering alerts. The Cybersecurity Service is also notified to ensure the system remains secure.
* **Evaluation:** This scenario examines how the system handles security events and ensures the safety of passengers. The responsiveness of the event bus and the coordination between security-related services are critical for success.

**Scenario 3: Adaptation to a New Vehicle Type**

* **Description:** The system is deployed in a new vehicle type (e.g., a heavy-duty truck) that has different sensor configurations and operational parameters.
* **Expected Behavior:** The Adaptability Service listens for events from the Vehicle Monitoring Service and adapts its configuration accordingly. The Configuration Management Service sends new profiles to the Vehicle Type Profiles Service.
* **Evaluation:** This scenario tests the system's adaptability and flexibility when supporting different vehicle types. It highlights the event-driven architecture’s ability to accommodate dynamic changes in vehicle profiles.

**2. Non-Functional Scenarios**

**Scenario 4: Scalability Under High Load**

* **Description:** A fleet of 10,000 vehicles is being monitored simultaneously, generating a high volume of events across all services. The system must scale to handle this load without performance degradation.
* **Expected Behavior:** The Event Bus efficiently manages the large volume of events, and individual services scale independently based on demand. No event loss or delays in processing should occur.
* **Evaluation:** This scenario focuses on the system's ability to scale horizontally and maintain performance under heavy load. The event-driven nature of the architecture must handle the high event throughput without bottlenecks.

**Scenario 5: Fault Tolerance and Recovery**

* **Description:** One of the services (e.g., Real-Time Processing Service) experiences a failure during operation. The system must continue functioning without interruptions, and the failed service must recover automatically.
* **Expected Behavior:** The Event Bus continues to route events to other services, and fault-tolerant mechanisms ensure that other critical services remain operational. The failed service recovers, and the system resumes normal operations.
* **Evaluation:** This scenario tests the fault tolerance and resilience of the architecture. The event-driven architecture’s decoupling of services ensures that failures in one component don’t bring down the entire system.

**Scenario 6: Security and Compliance**

* **Description:** The system must comply with strict cybersecurity regulations, including encryption of all communication and regular audits of security events. Additionally, any intrusion attempts should be automatically detected and responded to.
* **Expected Behavior:** The Cybersecurity Service ensures that all events are encrypted when transmitted across the Event Bus. The Logging Service records all security-related events for auditing, and the Intrusion Detection Service responds to any detected threats.
* **Evaluation:** This scenario evaluates the security aspects of the architecture, focusing on encryption, intrusion detection, and compliance with regulatory standards. The architecture’s ability to enforce security protocols and maintain compliance is crucial.

**Scenario 7: Low Power Consumption**

* **Description:** The system operates in electric vehicles, which have limited power resources. The system must minimize its energy consumption while maintaining full functionality.
* **Expected Behavior:** The Power Management Service optimizes energy usage by managing when and how different services are activated. Non-critical services reduce their energy consumption during low activity periods.
* **Evaluation:** This scenario examines the system’s energy efficiency and its ability to balance performance with low power consumption. The architecture's event-driven nature should allow services to become dormant when no relevant events are being processed, saving energy.

**Step 2: Document Scenario Interactions**

For each scenario, the following key interactions occur within the event-driven architecture:

1. **Event Generation:** A service (e.g., Vehicle Monitoring Service) generates an event based on a specific condition (e.g., sensor data threshold exceeded).
2. **Event Propagation:** The Event Bus distributes the event to all subscribed services that need to react to it.
3. **Event Handling:** Services that receive the event perform the necessary processing or take action (e.g., real-time processing, security response, configuration adjustment).
4. **Logging and Auditing:** The Logging Service records relevant events, particularly those related to security and compliance, for future auditing and diagnostics.

**Step 3: Prioritize Scenarios**

In a real-world application of SAAM, these scenarios would be prioritized based on their impact on system functionality and quality attributes. For example:

* **High Priority:** Real-Time Vehicle Monitoring, Security Threat Detection and Response, Fault Tolerance and Recovery.
* **Medium Priority:** Scalability Under High Load, Adaptation to New Vehicle Types, Security and Compliance.
* **Low Priority:** Low Power Consumption.

**Conclusion**

Applying SAAM to this event-driven architecture provides a structured way to evaluate how well the architecture meets both functional and non-functional requirements. The identified scenarios test various aspects of the system, such as real-time processing, scalability, fault tolerance, and security. Through these scenarios, stakeholders can assess the architecture’s strengths and weaknesses and make informed decisions about trade-offs and potential improvements.